



GRADO EN INGENIERÍA EN TECNOLOGÍAS DE LA
TELECOMUNICACIÓN

Curso Académico 2018/2019

Trabajo Fin de Grado

GAME OF WAR

Diseño y Desarrollo de una Aplicación Web con
HTML5 y CSS3

Autor : Nerea Cortés Lomana

Tutor : Dr. Gregorio Robles

Trabajo Fin de Grado

Game of War: Diseño y Desarrollo de una Aplicación Web con HTML5 y
CSS3

Autor : Nerea Cortés Lomana

Tutor : Dr. Gregorio Robles Martínez

La defensa del presente Proyecto Fin de Carrera se realizó el día de
de 2019, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 2019

*Dedicado a
mi familia*

Resumen

Este proyecto tiene como objetivo la creación de un juego de estrategia militar multijugador, en tiempo real y multilenguaje. Este juego debe ser accesible, por los jugadores, desde cualquier tipo de dispositivo con un navegador.

El diseño del juego está basado en el modelo cliente-servidor, de modo que ambos pueden estar alojados en la misma máquina, o en diferentes máquinas. Deberá soportar el acceso al mismo de múltiples jugadores al mismo tiempo.

Para desarrollar el cliente se ha utilizado el framework de JavaScript, Angular, y está programado en TypeScript. Se han utilizado también HTML5, CSS3 y Bootstrap. Para el servidor se ha utilizado Node.js y se ha desarrollado igualmente en TypeScript.

Summary

This project's aim is the development of a multiplayer military strategy game, in real time and multilanguage. This game must be accessible from any type of device with a browser.

The design is based on the client-server model, so that both can be hosted on the same machine, or on different machines. It must support multiple player access at the same time.

Angular, a JavaScript framework has been used to develop the client. And it has been programmed in TypeScript. HTML5, CSS3 and Bootstrap have also been used. Node.js has been used for the server and has also been developed in TypeScript.

Índice general

1. Introducción	1
1.1. Presentación	1
1.2. Estructura de la memoria	2
2. Objetivos	3
2.1. Objetivo general	3
2.2. Objetivos específicos	3
3. Estado del arte	5
3.1. Angular	5
3.1.1. Angular CLI	7
3.1.2. Ngx-tranlate	7
3.2. TypeScript	8
3.3. MySQL	8
3.4. Node.js	9
3.4.1. NPM	10
3.4.2. Express	10
3.4.3. Promise-MySQL	10
3.4.4. Jsonwebtoken	11
3.5. HTML5	12
3.6. CSS3	13
3.7. Bootstrap	13
3.8. Bcryptjs	13
3.9. Momentjs	14

3.10. Cors	14
4. Diseño e implementación	15
4.1. Arquitectura general	15
4.2. Diseño e implementación del servidor	16
4.2.1. Modelos de la base de datos	17
4.2.2. Estructura del servidor	22
4.2.3. API	25
4.3. Diseño e implementación del cliente	27
4.3.1. Uso de múltiples idiomas	29
4.3.2. Vistas de las páginas	30
4.3.3. Sistema de ataques	37
5. Resultados	43
6. Conclusiones	45
6.1. Aplicación de lo aprendido	45
6.2. Lecciones aprendidas	45
6.3. Trabajos futuros	46
A. Manual de instalación	49
Bibliografía	51

Índice de figuras

4.1. Arquitectura de la aplicación	15
4.2. Servidor	16
4.3. Base de datos	17
4.4. Tabla de Idiomas	18
4.5. Tablas de edificios	18
4.6. Tablas de tropas	19
4.7. Tablas de investigaciones	20
4.8. Tablas con información de usuarios	21
4.9. Esquema de ficheros del servidor	22
4.10. Funcionamiento autenticación de usuarios	24
4.11. Página de inicio	31
4.12. Página del perfil de un jugador	32
4.13. Página de edificios	33
4.14. Página de tropas	34
4.15. Página de investigaciones	35
4.16. Página de ataques	36
4.17. Informe	36
4.18. Página de informes	37
4.19. Reparto de soldados en un ataque	38
5.1. Diseño responsive 1	43
5.2. Diseño responsive 2	44

Capítulo 1

Introducción

1.1. Presentación

Esta memoria recoge el trabajo realizado para mi proyecto de fin de grado en ingeniería en tecnologías de la telecomunicación. El proyecto consiste en la realización de un juego de estrategia online.

Los primeros prototipos de videojuegos se remontan a la década de 1950. En la década de los 70 del siglo pasado aparecieron los primeros videojuegos dirigidos al gran público. Desde entonces la industria del videojuego ha pasado a convertirse en una de las mayores industrias del entretenimiento. El 54 % de la población Europea juega a videojuegos una media de 7,5 horas a la semana, según los estudios realizados por la ISFE¹ (Interactive Software Federation of Europe).

El videojuego es la primera opción de ocio de los españoles, más concretamente en 2018, España contaba con 15,8 millones de jugadores, según la AEVI² (Asociación española de videojuegos). Pero España sigue siendo más un país consumidor que dedicado a la producción y desarrollo de videojuegos.

Esto me ha llevado a comprobar el reto que supone la realización de un videojuego. El tipo de juego elegido es un juego de guerra de estrategia, en tiempo real, multijugador y online. Cada jugador se enfrentará a los demás jugadores. Cada uno de ellos dispondrá de una ciudad, la cual tendrá que desarrollar y proteger del resto.

¹<https://www.isfe.eu/games-in-society>

²<http://www.aevi.org.es/mas-15-millones-espanoles-juegan-videojuegos/>

El juego debe ser capaz de dar acceso a gran cantidad de usuarios y proporcionar soporte en múltiples idiomas. Para la realización se han escogido tecnologías que se han popularizado bastante estos últimos años, y que me gustaría poder llegar a conocer bien, como son Node.js y Angular.

1.2. Estructura de la memoria

La memoria está estructurada de la siguiente manera:

- **Capítulo 1. Introducción:** en este primer capítulo se realiza una descripción general del proyecto, especificando la estructura del mismo.
- **Capítulo 2. Objetivos:** se exponen los objetivos que se pretenden conseguir con la realización de este proyecto.
- **Capítulo 3. Estado del arte:** contiene una descripción de las tecnologías utilizadas para desarrollar la aplicación.
- **Capítulo 4. Diseño e implementación:** en este capítulo se explica el funcionamiento de la aplicación y cómo ha sido diseñada y desarrollada.
- **Capítulo 5. Resultados:** se realiza un análisis de los resultados obtenidos.
- **Capítulo 6. Conclusiones:** conclusiones tras la realización del proyecto.

Capítulo 2

Objetivos

2.1. Objetivo general

El objetivo de mi trabajo de fin de grado consiste en desarrollar un videojuego online de estrategia, en tiempo real y multijugador al que se pueda acceder a través de un navegador. Para ello utilizaremos distintas tecnologías de última generación que me permitan superar nuevos retos y mejorar mis habilidades y destrezas técnicas.

2.2. Objetivos específicos

Para alcanzar el objetivo principal se han seguido los siguientes objetivos específicos:

- Entretener a los jugadores de manera que disfruten dedicando tiempo al mismo, mientras viven una experiencia competitiva, a la vez que divertida.
- Funcionar en tiempo real: se pretende conseguir que el tiempo transcurra de manera continuada, estén activos o no los jugadores.
- Realizar un sistema de batalla equilibrado, basado en la diferencia de fuerzas de cada jugador, y que, además, para hacerlo menos previsible y más interesante, cuente con una serie de ventajas que al aplicarlas puedan influir en el resultado final.
- Proporcionar soporte en múltiples idiomas. La aplicación funcionará en español e inglés, teniendo la posibilidad de añadir nuevos idiomas.

- Desarrollar la aplicación de manera que se adapte a diferentes dispositivos o tamaños de pantalla.
- Contar con sencillez de uso: se espera que el juego sea fácilmente entendible por cualquier posible usuario.
- Usar tecnologías web avanzadas: JavaScript es uno de los lenguajes más populares de los últimos años. Y ha traído consigo numerosos *frameworks*, como Angular, el cual se pretende utilizar para el desarrollo junto con Node.js.

Capítulo 3

Estado del arte

Para el desarrollo del proyecto se han utilizado una serie de tecnologías ya existentes que se describen a continuación.

3.1. Angular

Angular [3] es un entorno de desarrollo de front-end que facilita la construcción de aplicaciones en HTML y TypeScript. Se trata de un framework de código abierto, creado por Google. Hay diferentes versiones de Angular, la primera de ellas es conocida como AngularJS y es totalmente diferente a las versiones posteriores, que se las conoce simplemente como Angular y que son mejoras de Angular 2.

Angular se utiliza para el desarrollo de aplicaciones web SPA, aplicaciones de una sola página (Single Page Application). Estas aplicaciones no recargan la página en ningún momento y la carga de datos es completamente asíncrona con el servidor. Este tipo de web son completamente dinámicas, el cambio entre las diferentes secciones de la web o entre las diferentes URLs de la web es prácticamente instantáneo, dinámico y reactivo. Las aplicaciones web SPA se han popularizado bastante estos últimos años y crearlas no es una tarea sencilla sin ningún framework, esto es una de las razones que han hecho a Angular tan popular. Es el entorno de desarrollo front-end más popular del mundo.

También es posible utilizar Angular para crear aplicaciones móviles, a través de otro complemento conocido como Ionic, y aplicaciones de escritorio.

La manera más estándar de programar con Angular es mediante el lenguaje TypeScript, que luego es compilado a JavaScript. La estructura de Angular se basa en 4 tipos principales de clases, que son simples clases con decoradores que marcan de que tipo son y proveen los metadatos necesarios para que Angular sepa cómo usarlas.

Módulos: En los módulos se especifican las dependencias de las aplicaciones, las clases que serán inyectables, los componentes que va a contener y cuál de ellos será el principal. Cada aplicación de Angular tiene un módulo raíz, llamado convencionalmente AppModule, que proporciona el mecanismo de arranque de la aplicación.

Al igual que los módulos de JavaScript, se pueden importar y exportar funcionalidades y permitir que sean utilizadas por otros módulos. De este modo, si se trata de aplicaciones muy grandes pueden tener varios módulos, uno para cada funcionalidad. Esta organización ayuda a gestionar el desarrollo de las aplicaciones complejas y a la reutilización. Además, de este modo se minimiza la cantidad de código que se carga al inicio, cargando los módulos bajo demanda.

Componentes: Un componente es el controlador de una vista, es decir, controla una parte de lo que aparece en pantalla.

Para convertir la clase en componente se usa el decorador @Component, en el que se especifica cómo se va a mostrar este componente utilizando un *template*, una plantilla HTML que define una vista, y para definir su comportamiento se utilizan los atributos de la clase y sus métodos. Los componentes pueden tener sub-componentes y cada aplicación Angular tiene al menos un componente, el componente raíz, que conecta de manera jerárquica el resto de componentes. Para que un componente se pueda comunicar con otros sub-componentes existen dos mecanismos, para enviar datos se utilizan propiedades y para recibirlos se utilizan eventos. Para implementar esa comunicación se usan *binding* (uniones).

Directivas: son clases que definen palabras clave a usar dentro de las plantillas. Cuando Angular las procesa transforma el DOM de acuerdo a lo especificado por las directivas. Pueden ser estructurales, si modifican el diseño, o de atributo, si modifican la apariencia o comportamiento de un componente.

Las plantillas combinan HTML con el marcado de Angular, como son las directivas y los *bindings*, uniones o enlaces entre los datos de la aplicación y el DOM.

Servicios: son clases que pueden ser usadas por los componentes para pedir datos u operaciones, se utilizan para aislar la lógica de la aplicación. Un ejemplo de uso es para hacer peticiones a APIs.

Se puede utilizar un mismo servicio dentro de múltiples componentes fácilmente gracias a la inyección de dependencias. La inyección de dependencias es un patrón de diseño en el cual los objetos se pasan por parámetro a una clase, en lugar de ser la clase la que instancia dichos objetos.

3.1.1. Angular CLI

Angular CLI [4] es una herramienta de línea de comandos utilizada para inicializar, desarrollar y mantener aplicaciones de Angular. Facilita mucho el proceso de inicio de cualquier aplicación con Angular, ya que mediante un solo comando puedes obtener todo el esqueleto de una aplicación. Al crear un proyecto genera también un archivo de configuración, cuyos valores se pueden modificar editando el archivo o por línea de comandos. También permite agregar nuevos componentes, servicios, directivas, etc.

Angular CLI incluye además un servidor, herramientas de compilación, *testing*, despliegue del proyecto, etc.

3.1.2. Ngx-translate

Es una librería de Angular [1] para la internacionalización de una aplicación. Permite definir traducciones para el contenido de la aplicación, en diferentes idiomas, y cargar dinámicamente estas traducciones, es decir, podemos cambiar de un idioma a otro sin necesidad de recargar la pantalla y de manera sencilla.

Ngx-translate pone a nuestra disposición un servicio de traducción, pipe y directivas, que nos ofrecen diferentes formas de realizar las traducciones, y métodos para establecer los idiomas disponibles, seleccionar el idioma por defecto de la aplicación, cambiar de idioma, etc.

Estas traducciones se guardan en archivos JSON y ofrecen la posibilidad de incluir variables en las traducciones.

Para utilizarlo es necesario importar los módulos y proveedores: TranslateLoader, Transla-

teModule y TranslateHttpLoader. Y configurar el módulo de traducción, TranslateModule, para utilizar un cargador personalizado. TranslateHttpLoader se encargará de la carga de los distintos archivos de idiomas desde src/assets/i18n/ utilizando HttpClient. También es posible cambiar la ruta y la extensión de archivo predeterminadas mediante su configuración.

3.2. TypeScript

TypeScript [14] un lenguaje de programación, de código abierto y desarrollado por Microsoft. Nació como una necesidad de mejorar algunos problemas de JavaScript, entre ellos el hecho de que es bastante complicado crear aplicaciones a gran escala con JavaScript. Está pensado para el desarrollo de aplicaciones robustas y escalables, y se puede utilizar tanto en el lado del cliente como en el del servidor junto a Node.js.

TypeScript es un superconjunto de JavaScript, es decir, se trata de un lenguaje basado en JavaScript. Los programas de JavaScript son programas válidos de TypeScript. Esto permite integrar JavaScript en proyectos ya existentes, sin tener que rehacer todo el proyecto en TypeScript, ya que todo el código TypeScript se compila en JavaScript nativo.

Algunas de las características de TypeScript son:

- Pone a nuestra disposición las herramientas de JavaScript ES6 (Ecmascript 6). De esta manera se mantiene a la vanguardia con las últimas mejoras de JavaScript.
- Cuenta con un tipado estático, es decir, al crear variables podemos añadir el tipo de dato, de esta forma es posible evitar errores en tiempo de ejecución.
- Objetos basados en clases, lo cual facilita mucho la programación orientada a objetos y añade más funcionalidad.

3.3. MySQL

Es un sistema de administración de base de datos relacional. MySQL [7] almacena los datos en tablas estructuradas, utiliza múltiples tablas conectadas por un campo en común, que hace posible combinar datos de las diferentes tablas, lo que supone mayor velocidad y flexibilidad.

Es un gestor multi-hilo, le permite manejar muchas tareas al mismo tiempo y ser utilizado por varias personas sin tener que esperar a que otros terminen sus consultas o procesos, lo que lo hace sumamente versátil.

MySQL cuenta con una licencia dual. Por una parte, es de código abierto (licencia pública general), y por otra, cuenta con una licencia comercial gestionada por Oracle Corporation.

Es muy utilizado en entornos de desarrollo web. Se la considera la base de datos de código abierto más popular y una de las más populares en general junto a Oracle y Microsoft SQL Server.

3.4. Node.js

Node [8] es un entorno de ejecución de JavaScript controlado por eventos asíncronos, diseñado para construir aplicaciones de red escalables.

Fue creado por Ryan Dahl en 2009 y está influenciado por sistemas como Event Machine de Ruby ó Twisted de Python.

Cuando se utiliza JavaScript en el lado del cliente, es el navegador quien, a través de un programa interno (un motor), interpreta el código JavaScript. De este mismo modo, Node.js proporciona un entorno de ejecución JavaScript del lado del servidor, haciendo uso del motor V8 de Google, que se encarga de interpretar el código y ejecutarlo.

El motor V8 es aquel que Google usa en su navegador Chrome y es posible descargarlo e incorporarlo a cualquier aplicación.

Node.js está pensado para soportar concurrencias muy altas gracias a su naturaleza asíncrona. El modelo de concurrencia más común consiste en la utilización de hilos del sistema operativo. Se genera un nuevo hilo para cada conexión, de modo que a mayor cantidad de personas, mayor cantidad de recursos consumidos del servidor y mayor número de servidores son necesarios. Node.js emplea un único hilo y un bucle de eventos asíncrono. Las nuevas peticiones son tratadas como eventos en este bucle, sin que se produzca ningún tipo de bloqueo en el flujo de trabajo.

3.4.1. NPM

Node Package Manager [9] es el administrador de paquetes de Node.js. Con él podemos instalar cualquier paquete Node.js que lleguemos a necesitar para nuestros proyectos, con una sola línea de código. Ayuda a administrar nuestros módulos, distribuir paquetes y agregar dependencias de forma sencilla. Está escrito enteramente en JavaScript y fue desarrollado por Isaac Z. Schlueter.

NPM utiliza el archivo `package.json` para almacenar los datos relevantes a nuestra aplicación. Este archivo corresponde a una estructura JSON, que incluye metadatos importantes para la descripción del módulo, la cual es utilizada en procesos tales como la instalación y publicación de los módulos (nombre, versión, autor, dependencias, etc.).

3.4.2. Express

Express [2] es el framework más popular de Node.js y muchos otros *frameworks* populares de Node.js están basados en él. Es rápido, minimalista y flexible, y proporciona todo lo necesario para crear tu servidor web sobre Node. Algunas características de Express son:

- Se comporta como un middleware para ayudar a administrar nuestros servidores y rutas.
- Amplia las funcionalidades del módulo HTTP que vienen por defecto con Node.js.
- Permite a los usuarios configurar rutas para enviar y recibir solicitudes entre el front-end y la base de datos.
- Viene con un sistema de visualización que admite más de 14 motores de plantillas.
- Permite establecer ajustes de aplicaciones web como qué puerto usar, la localización de las plantillas que se utilizan, etc.
- Proporciona un generador de aplicaciones que crea el esqueleto de una aplicación.

3.4.3. Promise-MySQL

El módulo Promise-MySQL sirve para conectar Node.js con MySQL, dando soporte para la utilización de promesas de JavaScript.

Se trata de un contenedor del controlador MySQLjs/MySQL de Node.js, que envuelve las llamadas con promesas Bluebird. Es decir, mientras el controlador MySQLjs/MySQL utiliza *callbacks*, *promise-mysql* soporta promesas.

3.4.4. Jsonwebtoken

Jsonwebtoken es una librería que permite generar y verificar JSON web tokens con Node.js.

Json web token(jwt) Jwt [13] : es un estándar abierto para crear tokens. Un token se utiliza para validar el acceso de un usuario a un servidor, se trata de una cadena alfanumérica que se obtiene al encriptar con una clave unos determinados datos que identifiquen al usuario.

Los jwt están compuestos por 3 partes separadas por puntos: header.payload.signature

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJsb2dnZWRJbkFzIjoiaWYWRtaW4iLCJpYXQiOiJlMjE0MjIzNzk2Mzh9.gzSraSYS8EXBxLNoWnFSRgCzcmJmMjLiuyu5CSpyHI
```

Header: La cabecera, contiene el algoritmo de encriptación (alg, comúnmente es el algoritmo HMAC SHA256) y el tipo de token (typ).

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Payload: Contiene atributos que definen nuestro token. Algunos de los atributos estándar de un token son:

- **Iat:** indica cuando fue creado.
- **Jti:** se trata de un identificador único para cada token.
- **Exp:** indica el tiempo de expiración del token y se utiliza una fecha para validar si el token ha expirado y obligar al usuario o volver a autenticarse.

Signature: La firma se genera utilizando las dos primeras partes, la cabecera y el contenido, en base64, y una clave secreta, key, que se utiliza para la encriptación.

```
signature = HMAC-SHA256(key, base64Encode(header) + '.' +
  base64Encode(payload))
```

3.5. HTML5

HTML [11] [12] (HyperText Markup Language) es un lenguaje de marcado que se utiliza para describir la estructura de las páginas web. HTML5 es la quinta versión de este lenguaje. A lo largo de todas las versiones se han incorporado y eliminado diferentes características, con el fin de hacerlo más eficiente y facilitar el desarrollo web compatibles con distintos navegadores y plataformas, y manteniéndose también compatible con las versiones anteriores.

A pesar de ser un estándar a cargo del Consorcio WWW, fue la asociación WHATWG, formada por Apple, Opera y Mozilla, la que publicó el primer borrador de HTML5 cuando W3C decidió dejar de evolucionar HTML. Más tarde W3C y WHATWG trabajaron juntos durante varios años en el desarrollo de HTML5, hasta que se separaron debido a sus diferentes objetivos, W3C quería publicar una versión terminada mientras que WHATWG quería seguir trabajando, manteniendo HTML5 en constante evolución.

Algunas de las principales características de HTML5 son:

- Incorpora nuevas etiquetas que especifican la semántica del contenido, ayudando a interpretar mejor la página. Donde antes teníamos `<div>` para definir cualquier sección, ahora podemos utilizar `<header>`, `<nav>`, `<section>`, `<footer>`, entre otros.
- Ofrece también mejoras en los formularios como validaciones, nuevos tipos de campos (email, date, range, etc) y nuevos atributos, por ejemplo, `multiple`, `placeholder`, `form`.
- Cuenta con numerosas APIs. Algunas APIs populares son:
 - Canvas: que permite hacer dibujos, juegos, animaciones, etc.
 - Geolocalización: permite mostrar e interactuar con un mapa de Google Maps.
 - Web Storage: permite almacenar datos del lado del cliente, cuando la sesión está activa.
 - Audio y video: permite incrustar elementos de audio y vídeo y reproducirlo desde el propio navegador.

3.6. CSS3

CSS [10] son las siglas de Cascading Style Sheets, es un lenguaje de diseño web utilizado para definir el aspecto visual de una página. Indica cómo tienen que aparecer los elementos en el navegador en cuanto a colores, tamaños de las fuentes, tamaños de elemento, tipo de letra, fondos de página, etc. Con CSS se consigue separar el contenido y el formato, dentro del mismo documento o separándolo en una hoja de estilo. Esto permite reutilizar una misma hoja de estilo en diferentes documentos y modificar por completo la estética de la página sin tener que modificar el contenido.

CSS3 es la última versión de este lenguaje, y proporciona nuevas características como bordes redondeados, sombras, gradientes, texto multi-columna, cajas flexibles, transiciones o animaciones, etc.

3.7. Bootstrap

Bootstrap [5] es un popular framework de desarrollo front-end. Facilita el diseño de páginas web y ofrece herramientas para crear de manera sencilla sitios y aplicaciones web con un diseño *responsive*, es decir, la interfaz se adapta de manera automática a cualquier dispositivo, tamaño y resolución de pantalla. Para ello utiliza un sistema de rejilla, compuesto por 12 columnas. Pone también a nuestra disposición elementos con estilos predefinidos fáciles de configurar como formularios, botones, cuadros, menús de navegación.

Bootstrap es modular y consiste en una serie de hojas de estilo LESS (un lenguaje de pre-proceso de CSS) combinado con JavaScript.

Originariamente se le conocía como Twitter Blueprint. Fue creado en 2010, para uso interno de la compañía, por Marck Otto y Jacob Thornton, trabajadores de Twitter, con el fin de paliar las inconsistencias producidas por el uso de varias librerías y el alto mantenimiento que ello suponía. Fue liberado con licencia MIT en el año 2011.

3.8. Bcryptjs

Bcryptjs es una librería de encriptación escrita en JavaScript por Niels Provos y David Mazières. Genera un hash combinando la información que se desea encriptar con un valor llamado

Salt, que son bytes aleatoriamente generados. De este modo al encriptar dos veces el mismo valor nunca generará el mismo hash. Esto evita los ataques Rainbow table y los ataques por fuerza bruta cuando se utiliza por ejemplo para guardar contraseñas en bases de datos.

Bcryptjs permite elegir el valor de saltRound, utilizado para calcular el campo Salt. Cuanto mayor sea este número más segura será la encriptación, pero mayor será el coste de procesamiento de los datos y más tiempo tarda en encriptar. Es decir, tenemos la seguridad frente a ataques contra la espera del usuario. Por defecto, el valor de saltRound es 10.

Ofrece también herramientas para comparar un valor con otro encriptado. Para realizar la comparación, en vez de desencriptar el campo encriptado, como hacen otros sistemas, Bcryptjs encripta el valor utilizando el valor Salt que va incorporado en el hash.

3.9. Momentjs

Momentjs [6] es una librería de JavaScript para gestionar fechas, más completa, sencilla y fácil de utilizar que el objeto Date de JavaScript. Moment js es un proyecto de código abierto, diseñado para ser utilizado tanto en un navegador como en Node.js. Permite analizar, validar, manipular y mostrar fechas en cualquier formato, etc. Además, tiene un robusto soporte internacional que permite mostrarlas en cualquier idioma.

3.10. Cors

Cors (Cross-origin resource sharing) es un mecanismo para permitir acceder a recursos de un servidor desde un origen distinto al que pertenece, lo cual no está permitido según la política de seguridad del mismo origen. Cors permite definir si es seguro permitir una petición de origen cruzado utilizando para ello cabeceras adicionales, antes de enviar la solicitud, se envía otra solicitud preliminar OPTION con la información del origen, y el servidor responde con los permisos de ese origen.

Una forma de utilizar CORS en un servidor de Express es mediante el módulo cors de npm.

Capítulo 4

Diseño e implementación

4.1. Arquitectura general

El proyecto consta de un cliente y un servidor independientes que interactúan entre sí. El cliente está desarrollado utilizando Angular 7. Está escrito en TypeScript y se utiliza además HTML, CSS3 y Bootstrap para la visualización de las vistas. Por otro lado, para el servidor se ha utilizado Node.js pero está igualmente programado en TypeScript. La aplicación requiere además de una base de datos, para la que se ha utilizado MySQL.

Para que el cliente y el servidor puedan comunicarse, es decir, para que el cliente pueda acceder a los recursos ofrecidos por el servidor es necesario la utilización de CORS, ya que se encuentran en dominios diferentes y, como se ha explicado antes, esto no está permitido por motivos de seguridad.

En la figura 4.1 podemos ver la arquitectura de la aplicación.

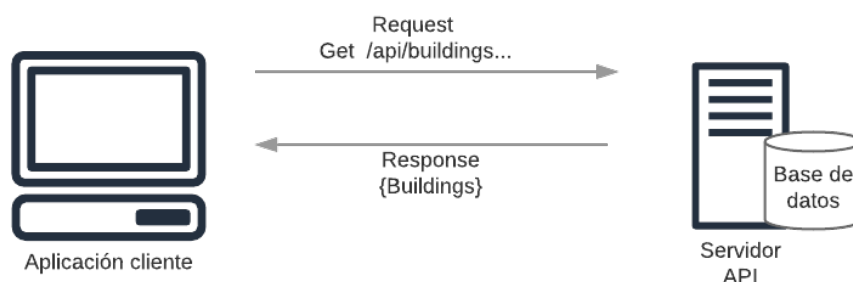


Figura 4.1: Arquitectura de la aplicación

Cuando un usuario interactúe con la aplicación cliente a través de un navegador, esta hará peticiones HTTP solicitando determinados recursos al servidor. El servidor accederá a la base de datos y responderá con un JSON, que el cliente tratará y mostrará. Por ejemplo, cuando un usuario acceda a la URL `/buildings`, el cliente tendrá que mostrar los edificios de ese usuario, de modo que hará una petición GET a la ruta `/api/buildings` y el servidor enviará un JSON con el modelo de datos de los edificios.

4.2. Diseño e implementación del servidor

Se ha creado un API REST para ofrecer los datos del juego. El servidor ofrece una serie de recursos, que veremos más adelante, y se encarga de los accesos a la base de datos, en función de las peticiones que reciba del cliente.

El servidor, realizado con Node.js, gestiona eventos de manera asíncrona, como se ha explicado. Existen diferentes modelos de programación asíncrona. El modelo más común es la utilización de *callbacks*, es decir, cada operación asíncrona recibe una función que incluye lo que debe hacer a continuación. Pero este método dificulta la comprensión del código.

El método elegido para realizar esta programación asíncrona ha sido el uso de promesas junto con los operadores `async` y `await`. De este modo el esquema de desarrollo se parece algo más al estilo secuencial pero manteniendo su carácter no bloqueante.

`Async/await` nos permite ejecutar las promesas y esperar el resultado de forma asíncrona, sin la necesidad de los bloques `then` y `catch`.

De este modo cada acceso a la base de datos y la comunicación con el cliente se hace mediante promesas.

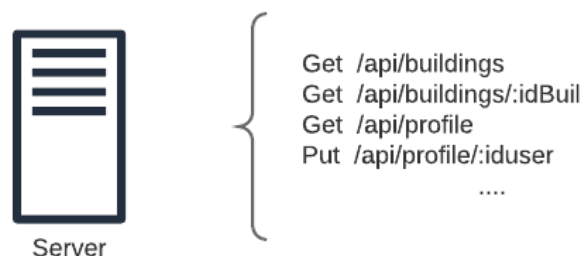


Figura 4.2: Servidor

4.2.1. Modelos de la base de datos

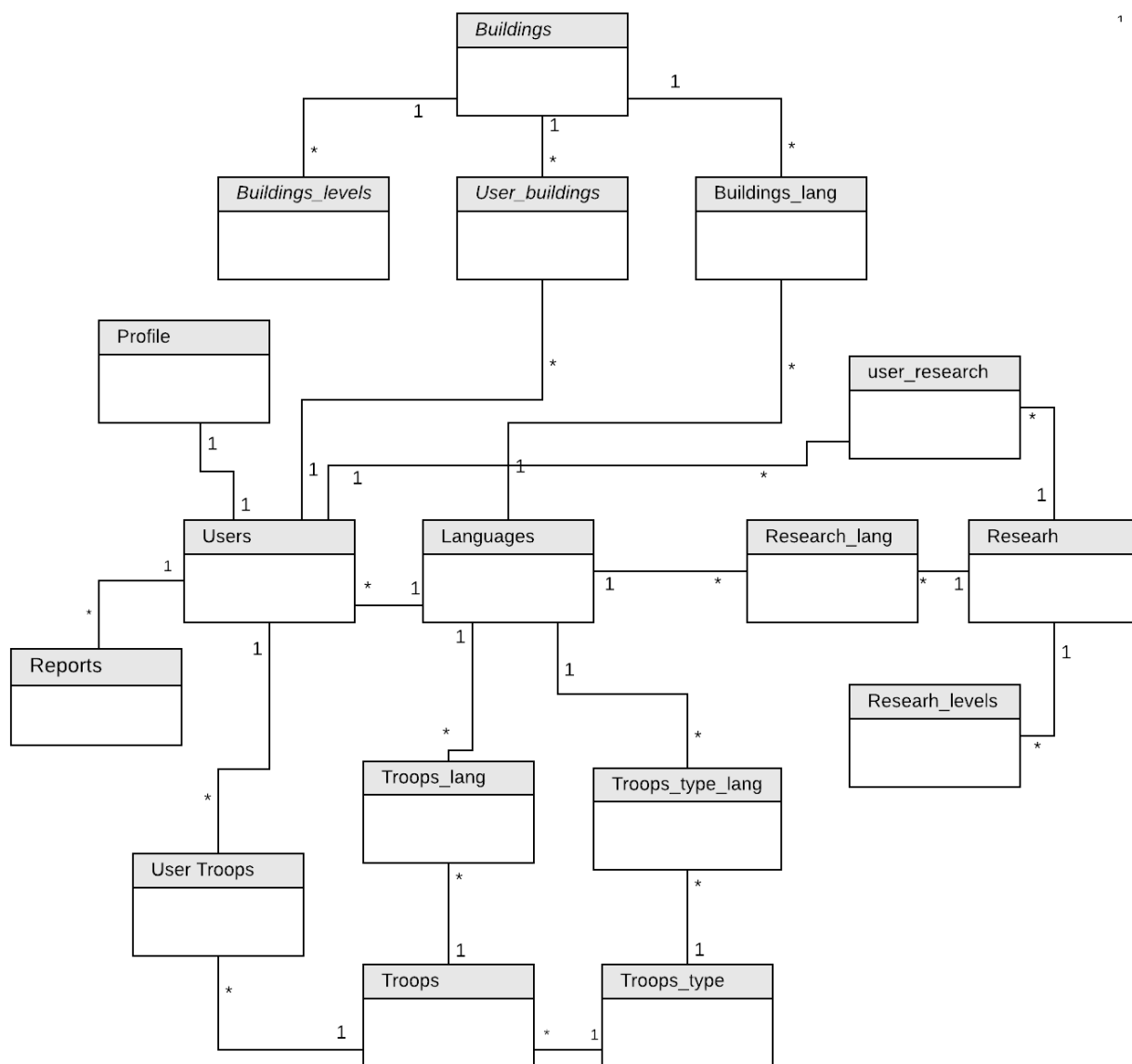


Figura 4.3: Base de datos

La aplicación requiere de una base de datos. Parte de las tablas están rellenas inicialmente con los datos del juego.

La aplicación es multilenguaje, español e inglés, de modo que la información almacenada que deba ser traducida estará almacenada en los dos idiomas. El resto de las tablas, pertenecientes a los datos de los usuarios se irán rellorando al crearse nuevos usuarios.

Todo esto está definido dentro del archivo database.sql.

El esquema global de la base de datos se muestra en la figura 4.3.

En las siguientes figuras se mostrarán las tablas separadas para poder verlas más detalladamente.

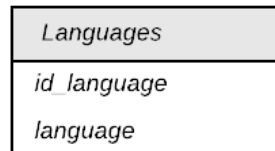


Figura 4.4: Tabla de Idiomas

La tabla *languages* almacena los diferentes idiomas en los que es posible mostrar la aplicación. Como veremos más adelante sería sencillo añadir nuevos idiomas. Al igual que nuevos tipos de edificios e investigaciones con las utilidades que aportan, o tropas, ya que solo habría que modificar las tablas para añadir el contenido, no habría que tocar el código.

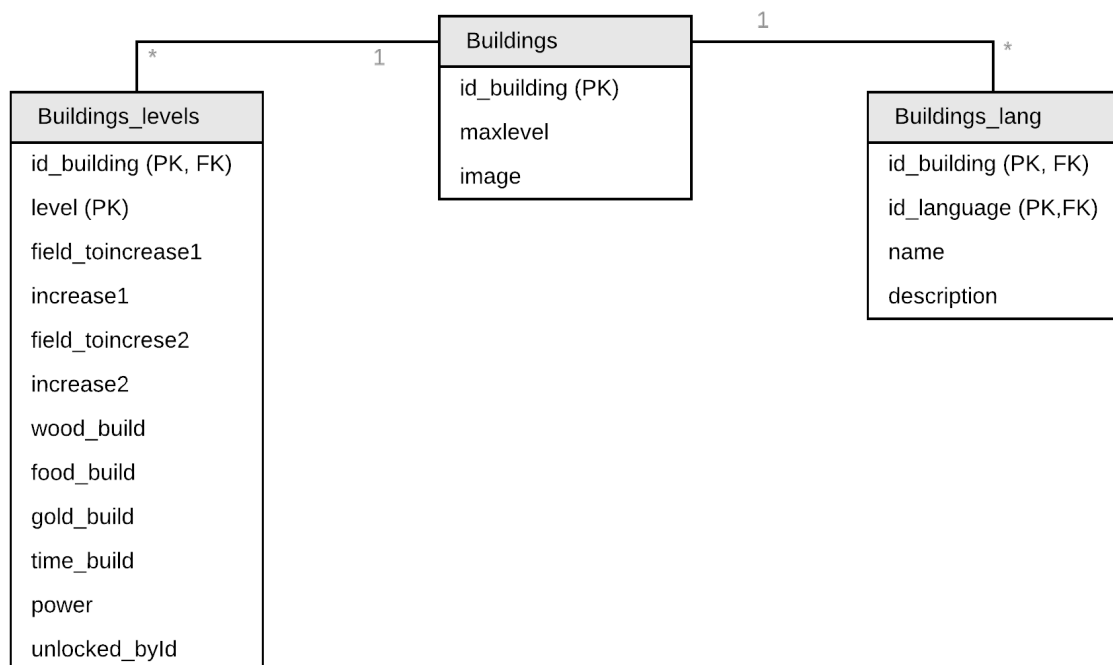


Figura 4.5: Tablas de edificios

En la figura 4.5 se muestran las tablas correspondientes a la información de los edificios. Cada edificio existente está guardado en la tabla *buildings* y es identificado con el campo *id_building*, que es clave foránea de las otras dos tablas.

La información que debe ser traducida de un edificio está almacenada en la tabla *buildings_land*. Cada edificio, con un mismo *id_building*, es almacenado en los idiomas existentes en la tabla *languages*, de la cual es clave foránea el campo *id_languages*.

Los edificios tiene diferentes niveles y cada uno de estos niveles está especificado en la tabla *buildings_levels*: se detallan los recursos y el tiempo necesarios para mejorarlo, las mejoras que se producen al subirlo de nivel (*field_toIncrease1* y *field_toIncrease2*), etc.

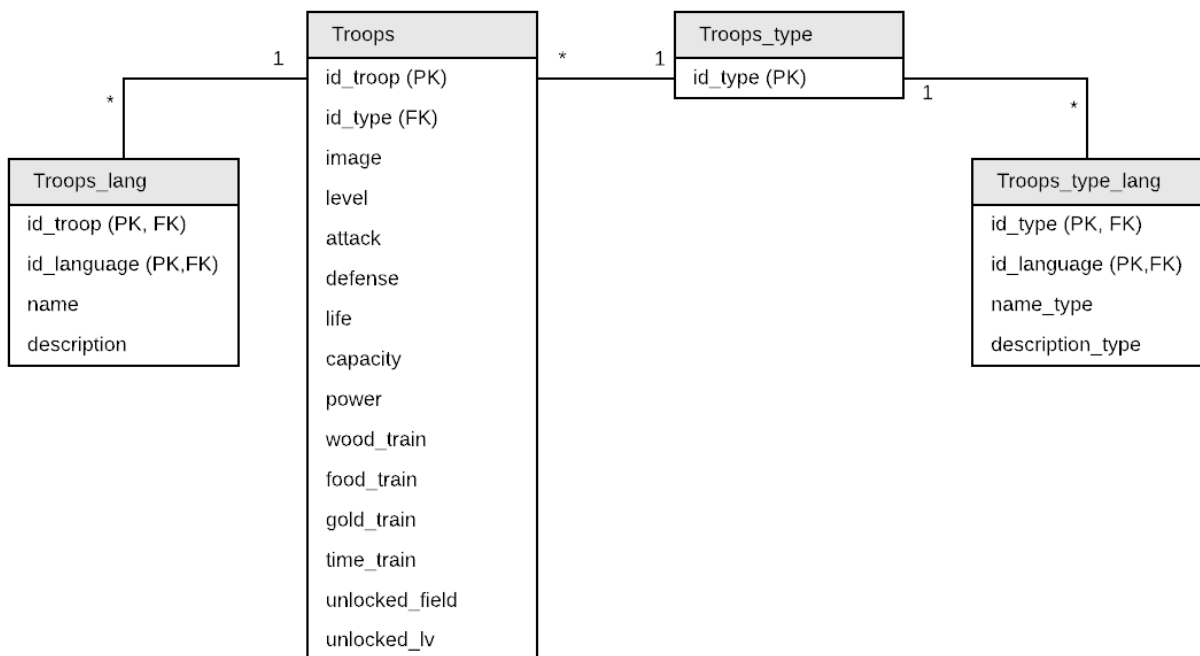


Figura 4.6: Tablas de tropas

En la tabla *troops_type* de la figura 4.6 están almacenados los tipos de tropas en los que pueden clasificarse todas las tropas existentes, almacenadas en la tabla *troops*. De este modo, *id_type* es la clave primaria de la tabla *troops_type* y clave foránea de la tabla *troops* ya que es utilizado para hacer referencia a la tabla *troops_type*.

En la tabla *troops_type_land* tenemos, nuevamente, el nombre y la descripción según el idioma.

En la tabla *troops* tenemos las tropas identificadas con el campo *id_troops* y se indica tam-

bién: el nivel (dentro de cada tipo de tropa estas pueden ser más o menos poderosas); los recursos y el tiempo necesarios para crearlas; el ataque, defensa, vida y capacidad de carga de recursos de las tropas; qué edificio permitirá crear cada tipo de tropas (`unlocked_field`) y a qué nivel (`unlocked_lv`). La tabla *troops_land* es su correspondiente tabla de idiomas, y como puede verse hace referencia a ella mediante el campo `id_troop` y a la tabla de idiomas mediante el campo `id_language`.

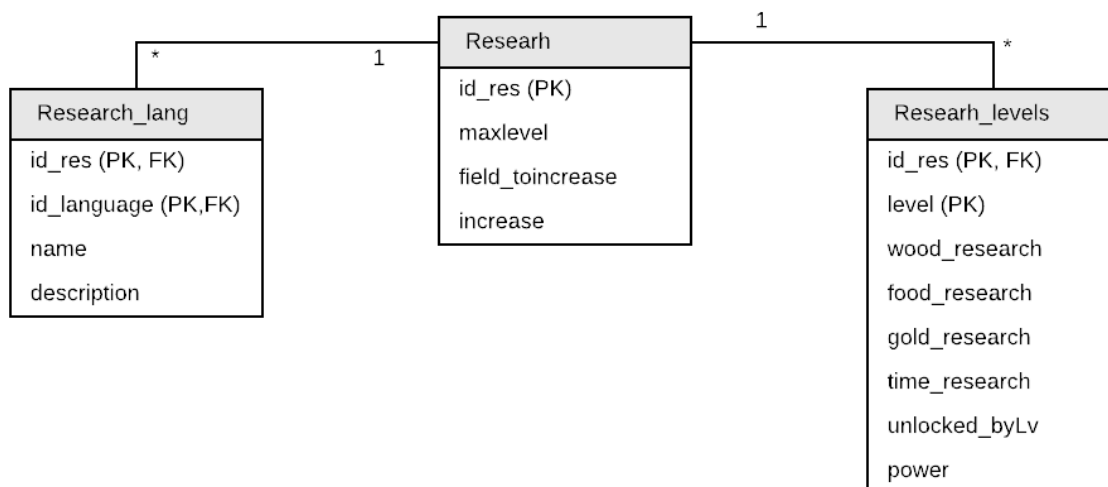


Figura 4.7: Tablas de investigaciones

En la figura 4.7 se muestran las tablas correspondientes a la información de las investigaciones.

Cada investigación que se puede realizar está guardada en la tabla *research* y se identifica mediante el campo `id_res` (clave primaria). Se indica también el nivel máximo que puede alcanzar una investigación y qué es lo se consigue al mejorar su nivel.

La información que debe ser traducida de la tabla *research* está almacenada en la tabla *research_land* y nuevamente vemos como hace referencia a ella y a la tabla *languages*.

Cada nivel está especificado en la tabla *research_levels*: los recursos y el tiempo necesarios para mejorar el edificio, cuándo estará desbloqueada la investigación para poder mejorarla y el poder que proporcionará. La clave de la tabla está formada por los campos `id_res` y `level`.

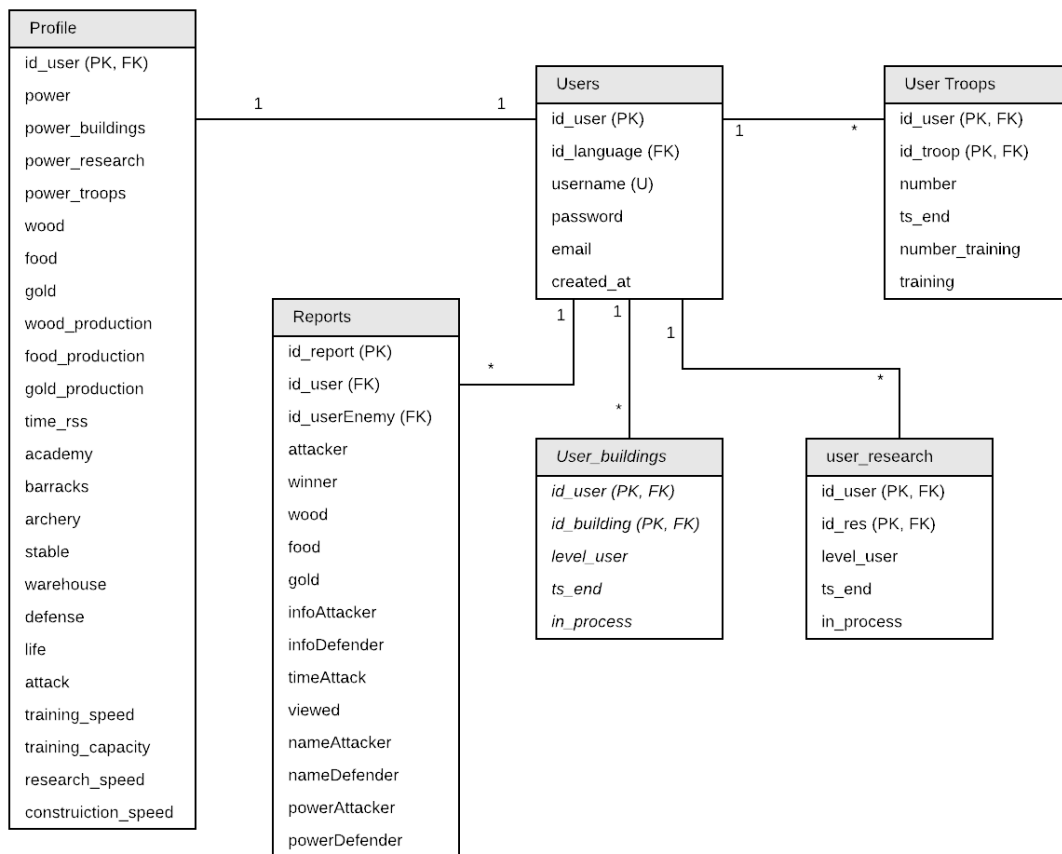


Figura 4.8: Tablas con información de usuarios

Cuando se registra un nuevo jugador se crea un nuevo registro en cada una de las tablas de la figura 4.8, excepto en la tabla *reports*.

La tabla *users* contiene a todos los jugadores. El campo *id_user* es la clave única de la tabla. El nombre de usuario es único, de modo que solo puede haber un jugador con un mismo nombre. El correo puede repetirse, pero podría utilizarse para limitar el número de cuentas de los jugadores. La contraseña se encripta antes de almacenarla en la base de datos para mayor seguridad.

Las tablas *users_buildings*, *users_troops* y *users_research* almacenan la información de cada edificio, tropa o investigación del usuario: el nivel en actual, si hay algún proceso (construcción, investigación o creación de tropas) en marcha, el tiempo que queda para finalizar ese proceso, etc.

La información general de cada partida se guarda en la tabla *profile*: los recursos actuales del jugador, la producción de recursos, el poder, los *bonus* obtenidos de velocidad de construcción,

investigación y creación de tropas, los *bonus* de ataque, defensa y vida, etc.

La tabla *reports* se utiliza para almacenar la información de los ataques realizados y recibidos de cada jugador. Por cada ataque se crean dos registros, uno para cada jugador, y de esta manera se le podría dar mayor control sobre su informe a cada uno. Los campos almacenados son, por ejemplo, la hora a la que se produjo el ataque, el ganador, los recursos perdidos o ganados, la información relativa a las tropas que participaron en la batalla, etc.

4.2.2. Estructura del servidor

El servidor está estructurado, de manera sencilla, como se muestra en la imagen 4.9.

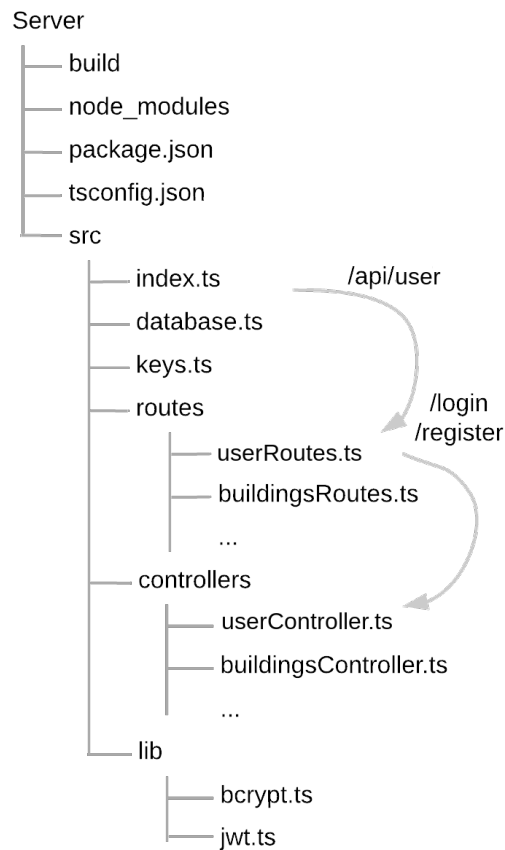


Figura 4.9: Esquema de ficheros del servidor

El archivo de Node llamado **package.json** contiene principalmente el nombre, la descripción, una sección de dependencias donde aparecen los módulos que he utilizado, y que he des-

crita en el capítulo 3, y una sección de scripts donde podemos crear nuestros propios comandos.

El archivo **tsconfig.json** es el archivo de configuración de TypeScript. En él se especifica, por ejemplo, a qué versión de JavaScript tiene que traducirse TypeScript. En mi caso ECMAScript 6, “target”: “es6”. También se puede especificar en qué directorio tiene que colocar ese código traducido a JavaScript. Yo he creado una carpeta llamada build, dentro de mi servidor, y es ahí donde quiero que coloque este código: “outDir”: “./build”.

En el archivo **database.ts** se establece la conexión de la base de datos, de esta manera no tenemos que establecerla todo el tiempo. Cuando se quiere utilizar una consulta en otra parte del código simplemente se llama al módulo y este ya tiene la conexión. Para establecer la conexión se pasan los parámetros de la base de datos: en qué *host* está instalado, el usuario, la contraseña, en que puerto está instalado, etc. Estos parámetros están definidos en el archivo **keys.ts**.

El archivo **index.ts** es el archivo principal de la aplicación. En él configuramos el servidor de Express: se le asigna un puerto (en el caso de que haya un puerto definido en el sistema será el que utilice), se habilita el uso de JSON en las peticiones HTTP, se configura CORS (esta configuración se ha hecho de tal manera que solo acepte peticiones del origen de mi cliente), se establecen las rutas y se inicia el servidor.

Dentro de la carpeta **routes** tenemos todas las rutas organizadas, de manera que todas las rutas relacionadas están dentro de un mismo enrutador. Y del mismo modo, dentro de la carpeta **controllers** tenemos una clase para cada clase de routes, donde se especifica que tiene que hacer ese conjunto de rutas relacionadas. Es decir, por ejemplo, las rutas relativas a la autenticación de usuarios (/api/user/...) están dentro de userRoutes.ts y en userControllers.ts está el proceso que realiza de cada una de esas rutas.

Dentro de la carpeta **lib** tenemos dos clases: **jwt.ts**, con métodos para la creación y validación de tokens, y **bcrypts.ts**, con métodos para la encriptación y validación de las contraseñas.

Encriptación de la contraseña

La validación y encriptación de la contraseña se realiza mediante el uso del módulo Bcryptjs. Se utiliza la función asíncrona *hash* para encriptar la contraseña, con un genSalt, explicado en el capítulo anterior, de 10. Para validar una contraseña introducida por un usuario se utiliza la

función asíncrona *compare*, que compara la contraseña introducida con el hash almacenado en la base de datos.

Funcionamiento autenticación de usuarios

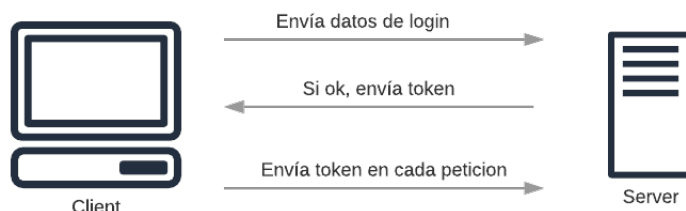


Figura 4.10: Funcionamiento autenticación de usuarios

- El cliente envía los datos del usuario al servidor, recabados a partir de la página de login/registro.
- El servidor recibe los datos del usuario. En el caso de un usuario nuevo lo da de alta y genera un token. En el caso de un usuario ya registrado comprueba que el nombre de usuario y la contraseña se corresponden con los almacenados y genera un token. En ambos casos el token generado se envía de vuelta al cliente. En caso de que el registro o el login no fuera correcto, el servidor envía un código de error 401 al cliente.
- Si la autenticación ha sido correcta, el usuario recibe el token y lo almacena en el almacenamiento local del navegador (LocalStorage) para poder disponer de él en las futuras peticiones que haga al servidor. De este modo si se refresca la página o se accede a ella pasado poco tiempo se seguirá disponiendo del token generado y se evita que el usuario tenga que repetir el proceso de login.
- Cada vez que el cliente tenga que acceder de nuevo al servidor se recoge el token del LocalStorage y se añade a la cabecera de la petición HTTP.
- El servidor antes de acceder al recurso solicitado, extraerá el token de la cabecera de la petición y lo validará.
- Cuando el token es inválido y el cliente recibe un código de error, este lo elimina del LocalStorage e indica al usuario que no está logueado.

Generación y validación del token

Para generar el token utilizado y validarlo he utilizado el módulo `Jsonwebtoken`. Como se ha comentado en el capítulo anterior, para generar un `jwt` se utiliza una clave para la encriptación, la cual está definida en el servidor, y un `payload` con los atributos del token. El `payload` que he utilizado es:

```
var payload = {
  username: user.username,
  id: user.id_user,
  id_language: user.id_language
}
```

El token creado expira en 24 horas.

He incluido en el token el `id` de usuario para utilizarlo en las peticiones. También estoy utilizando el idioma para recoger solo los datos del idioma seleccionado por el usuario.

Para validar el token se extrae de la cabecera `textitAuthorization` de la petición `HTTP` y se valida utilizando la función `verify` de `jwt`. Si el token no iba en la cabecera, si la sesión ya ha expirado o es un token inválido, el servidor devuelve un error `401` al cliente. Si es válido continua con la petición del cliente.

4.2.3. API

Operaciones y recursos disponibles de la API:

- **post api/user/login** : comprueba que el nombre de usuario y la contraseña recibidos se corresponden con los almacenados accediendo a la tabla `users`. Si es así genera un token y lo envía al cliente. En caso de que el usuario no exista o la contraseña sea incorrecta devuelve un error `HTTP 401`.
- **post api/user/register** : encripta la contraseña e inserta un registro en la tabla `users` junto con el resto de datos introducidos por el usuario. Crea también un registro en las tablas `user_buildings`, `user_troops`, `user_research` y `profile`, con los datos iniciales de la nueva partida.
- **get api/user/** : realiza una `select` de la tabla `users` y devuelve el nombre y el idioma del usuario.

- **put api/user/** : modifica el idioma de un usuario: actualiza el campo `id_language` de la *users* del usuario que realiza la petición.
- **get api/language/** : devuelve los idiomas disponibles, es decir, realiza una select de la tabla *languages* y devuelve el contenido.
- **get api/profile/** : devuelve el perfil del jugador.
- **get api/profile/enemyList** : recupera una lista de 30 usuarios de la tabla *profile*. Para hacer la select de esa lista primero se generan identificadores de usuarios que necesitará para la búsqueda, de la siguiente manera: Se recuperan de la tabla *reports* los usuarios que han atacado al jugador que realiza la petición, si el número total de estos es inferior a 30, se complementa la lista con identificadores de usuario generados aleatoriamente.
- **put api/profile/:idUser** : actualiza la partida del jugador con el identificador `idUser`, la tabla *profile*.
- **get api/research/** : recupera los datos relacionados de las tablas *user_research*, *research*, *research_lang* y *research_levels*, es decir, la información del jugador en cuanto a investigaciones, y la devuelve.
- **get api/research/timeToEnd** : devuelve el tiempo restante para que finalice la investigación que haya en curso.
- **put api/research/:idRes** : actualiza la tabla *user_research* para la investigación con el identificador indicado, `idRes`.
- **get api/buildings/** : recupera los datos relacionados de las tablas *user_buildings*, *buildings*, *buildings_lang* y *buildings_levels*, es decir, la información de un jugador relativa a los edificios.
- **get api/buildings/timeToEnd** : recupera el tiempo restante para que finalice el edificio que haya en construcción.
- **get api/buildings/:idBuil** : devuelve la información de un edificio concreto, `idBuild`, relativa al jugador.

- **put api/buildings/:idBuil** : actualiza la tabla *user_buildings* para el edificio con el identificador indicado, *idBuil*.
- **get api/troops/** : devuelve la información de las tropas (*troops* y *troops_lang*).
- **get api/troops/user/timeToEnd** : devuelve el tiempo restante para que finalicen las tropas que haya creandose.
- **get api/troops/user/all** : devuelve la información de las tropas relacionadas con el jugador que realiza la petición. Es decir, los datos relacionados de las tablas *user_troops*, *troops*, *troops_lang*, *troops_type* y *troops_type_lang*.
- **get api/troops/user/:idUser** : devuelve la información de las tropas relacionadas con el jugador con el identificador *idUser*, de las tablas *user_troops*, *troops* y *troops_lang*.
- **put api/troops/:idTroop** : actualiza la tabla *user_troops* para las tropas con el identificador *idTroop* y el usuario que realiza la petición.
- **put api/troops/number/:idUser** : actualiza el número de tropas de un jugador: el campo *number* de la tabla *user_troops* para todas las tropas del usuario *idUser*.
- **get api/reports/** : devuelve los informes de ataque del jugador que realiza la petición.
- **put api/reports/:idReport** : actualiza el contenido de un informe de ataque, identificado con *idReport*. Tabla *reports*.
- **post api/reports/** : Crea un registro, un informe de ataque, en la tabla *reports*.

Todas las respuestas generadas tienen formato JSON.

4.3. Diseño e implementación del cliente

El cliente es el encargado de interactuar con los usuarios a través de un navegador web. Tal y como he explicado anteriormente, la parte del cliente está desarrollada utilizando Angular 7. Angular proporciona un servidor en el puerto 4200, que es el encargado de atender a las peticiones del usuario.

Al tratarse de una aplicación SPA no se recarga la página en ningún momento. La carga de

datos es completamente asíncrona con el servidor y, además, este tipo de web es completamente dinámica, es decir, el cambio entre las diferentes urls de la web es completamente instantáneo, dinámico y reactivo.

Angular se compone de componentes, servicios, módulos y directivas. La estructura general del cliente es la siguiente:

- **index.html** es la página HTML inicial del proyecto, aquí tenemos la cabecera de HTML con el cuerpo que llama directamente al componente raíz `app.component`.
- **main.ts** es el archivo TypeScript inicial, en el se establecen los módulos necesarios, en nuestro caso `app.module.ts`.
- **styles.css** en el se establecen los estilos globales de la aplicación.
- **app.module.ts** es el módulo principal de mi proyecto, en él se declaran los componentes, los proveedores y los módulos que se utilizan.
- **app.component.ts** es el componente raíz y su plantilla **app.component.html** divide la página de manera que la parte de arriba muestra siempre una vista controlada por el componente `navigation` y el resto de la página varía. Esta parte que varía es controlada por un Controlador diferente según la ruta. Estas rutas están especificadas en la clase **app-routing.module.ts**.
- Dentro de la carpeta **components** tenemos cada uno de los componentes (como es el caso de `login.component.ts`) con las plantillas que controlan (`login.component.html`) y los estilos de esa plantilla (`login.component.css`). Los componentes `building`, `research` y `troops`, son hijos de los componentes `buildings-list`, `research-list` y `troops-list` respectivamente. Es decir, estos componentes están completamente relacionados y se comunican la información constantemente entre ellos.
- Dentro de **services** tenemos todos los servicios utilizados, que pueden ser llamados desde cualquier componente. Estos servicios hacen peticiones al servidor para obtener o modificar datos de la base de datos.

- Dentro de la carpeta **i18n** tenemos diferentes JSON utilizados para el sistema de internacionalización de la aplicación. La forma en que se utilizan estos JSON se explicará en la siguiente sección.
- Las clases **interceptor.ts** e **interceptorError.ts** interceptan las peticiones entre el cliente y el servidor. **interceptor.ts** añade el token de autorización en las cabeceras de las peticiones al servidor y **interceptorError** borra el token del LocalStorage del navegador (cierra la sesión del usuario) y muestra la vista de login, caso de que el cliente reciba un error de autenticación.
- Al igual que en el servidor, tenemos una carpeta llamada **node_modules**, que contiene todas las dependencias instaladas, y un archivo llamado **package.json**, que contiene la descripción de esas dependencias. Esto es así ya que Angular también utiliza Node para poder ejecutar el servidor y para poder convertir los archivos de TypeScript. En el archivo **tsconfig.json** tenemos la configuración para la compilación de archivos de TypeScript.

4.3.1. Uso de múltiples idiomas

Como se ha dicho, dentro de la carpeta **i18n** tenemos un JSON para cada idioma en el que se quiera mostrar la aplicación. Estos JSON incluyen todo el contenido de las vistas que quiera ser traducido, y todos ellos deben tener el mismo contenido. Un ejemplo de uso es el siguiente:

Archivo `es.json`:

```
{
  "profile": {
    "profile": "Perfil",
    "economy": "Economía",
    ...
  }, "detail": {
    "errorLockById": "Necesario nivel {{level}} de {{name}}",
    "errorNotRss": "Recursos insuficientes",
    ...
  }
  ...
}
```

Archivo en.json:

```
{
  "profile": {
    "profile": "Profile",
    "economy": "Economy",
    ...
  }, "detail": {
    "errorLockById": "Required level {{level}} of {{name}}",
    "errorNotRss": "Insufficient resources",
    ...
  }
  ...
}
```

Las palabras entre corchetes `{{}}` son variables que pueden enviarse desde las plantillas. El modo de utilizar estos JSON es mediante la librería `Ngx-translate`, de internacionalización para Angular. Dentro del servicio `languages.service.ts` tenemos los métodos para establecer un lenguaje por defecto y modificar el lenguaje, los cuales hacen uso del servicio `TranslateService`, inyectado como dependencia desde `@ngx-translate/core`.

El lenguaje de un usuario se extrae del almacenamiento local del navegador, donde es almacenado al iniciar la sesión, y se establece mediante las funciones anteriores. Para que las vistas se muestren en un idioma u otro se utilizan las directivas de traducción.

Incluir un nuevo idioma sería sencillo, no habría que modificar código. Solo habría que añadir un nuevo JSON con las traducciones y añadir nuevos registros en las bases de datos para incluirlo.

4.3.2. Vistas de las páginas

A continuación se describen las páginas que componen la aplicación. Para utilizar la aplicación es necesario estar registrado, de modo que si no se ha iniciado sesión solo se podrá acceder a la página de Login/Registro. Para mejorar la visualización de las páginas he utilizado Bootstrap y CSS3. Con Bootstrap se consigue además una correcta visualización para diferentes tamaños de pantalla y dispositivos.

Página de inicio

La aplicación parte de una página de login/registro. Se muestra un formulario para iniciar sesión o registrarse. Mediante un botón en la parte de abajo del formulario se puede elegir entre una opción u otra.

Se ha utilizado un formulario reactivo, sin enlace con los datos, que deja la lógica del formulario del lado del controlador. De este modo en el controlador se declara el formulario como objeto para tener control sobre su valor y su estado. A cada uno de los elementos del formulario se les asignan valores y métodos de validación, y se asocian al formulario. Estos métodos de validación comprueban si son campos obligatorios, si la longitud es correcta, si el formato del email es correcto, etc.

La parte de arriba de todas las páginas está controlada por el componente navigation, en este caso solamente se muestra una barra superior con el título de la aplicación, ya que el usuario no ha iniciado sesión aún.

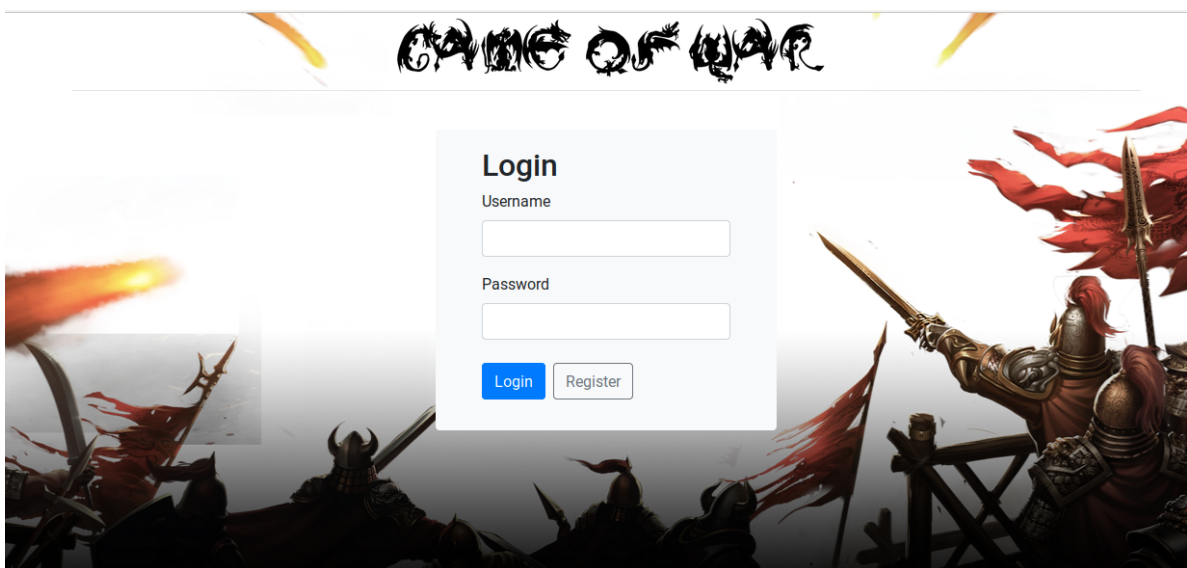


Figura 4.11: Página de inicio

Página de perfil del jugador

Una vez que inicias sesión se redirige a la ruta /profile. Ahora podemos ver que la barra de navegación ha cambiado. En el lado derecho se muestra un botón para cerrar la sesión, Logout. En el lado izquierdo información de la partida del usuario: poder y los recursos que tiene. Además, debajo de la barra se ha añadido un menú con 6 opciones. El perfil del usuario,

la página donde estamos, es la primera de esas opciones.

El resto de la página pasa a ser controlada por el componente profile. Y aparece información de la partida dividida en 3 pestañas. En la primera pestaña, perfil, se muestran datos del usuario: el nombre, el poder, el idioma elegido, etc. Tenemos también un botón de configuración que permite modificar el idioma elegido: al pinchar en el botón aparece una modal de Bootstrap con un campo select para elegir el idioma.

En la segunda pestaña se muestran las estadísticas y mejoras relacionadas con la parte del juego económica, más adelante podremos ver de dónde salen estas mejoras: la producción de recursos por hora, la velocidad de construcción, etc. En la tercera pestaña se muestran las estadísticas y mejoras relacionadas con la parte militar, aquello relacionado con las tropas, el ataque y la defensa.

Mejoras económicas	
Producción de madera	0/h
Producción de comida	1000/h
Producción de oro	1000/h
Protección de recursos del almacén	350000
Velocidad de construcción	+2.5%
Velocidad de investigación	+0%

Figura 4.12: Página del perfil de un jugador

Página de edificios

Al pulsar en el menú sobre la opción Edificios se redirige a la ruta /buildings. Podemos ver como la parte de arriba de la página se mantiene, la navegación, que como se ha dicho, es fija.

Ahora el resto de la página pasa a ser controlado por el componente buildings-list, que a su vez divide la página. El lado derecho queda entonces controlado por su hijo, el componente buildings.

Podemos ver, en el lado izquierdo, una lista con todos los edificios disponibles. Cada edificio tiene una funcionalidad diferente, permite crear soldados, genera recursos, etc. Al pinchar en un edificio de la lista nos lo muestra más detalladamente: vemos para qué sirve el edificio, si se puede o no subir de nivel, las mejoras que supondría subirlo de nivel, el coste en recursos y tiempo.

También es posible que no pudiera mejorarse el edificio, bien porque ya ha alcanzado el máximo nivel, o porque no tenemos recursos suficientes, o quizá porque primero es necesario alcanzar cierto nivel con otros edificios. Además, solo es posible poner en construcción un único edificio a la vez. Cuando se está construyendo aparece un mensaje en la parte superior de la lista con el edificio en cuestión y una cuenta atrás para su finalización. Esta cuenta atrás aparece también dentro del detalle del edificio.



Figura 4.13: Página de edificios

Página de tropas

La tercera opción del menú, Tropas, redirige a la ruta /troops. El componente troops-list se encarga de mostrar un desplegable en el lado derecho de la pantalla, con los tipos de tropas que existen (Infantería, Caballería y Arqueros) y al desplegar podemos ver todas las tropas

disponibles de cada tipo.

En el lado derecho, controlado por el componente `troops`, aparece el detalle de una tropa, una vez que se ha pulsado sobre ella. Para cada tipo de tropas existe un edificio que permite crearlas y, dependiendo del nivel del edificio, solo se podrán crear ciertas tropas de ese tipo. Cuando haya tropas entrenándose aparecerá, en el propio desplegable con la lista y dentro del detalle la información, el tiempo restante y el número de tropas que hay creándose.

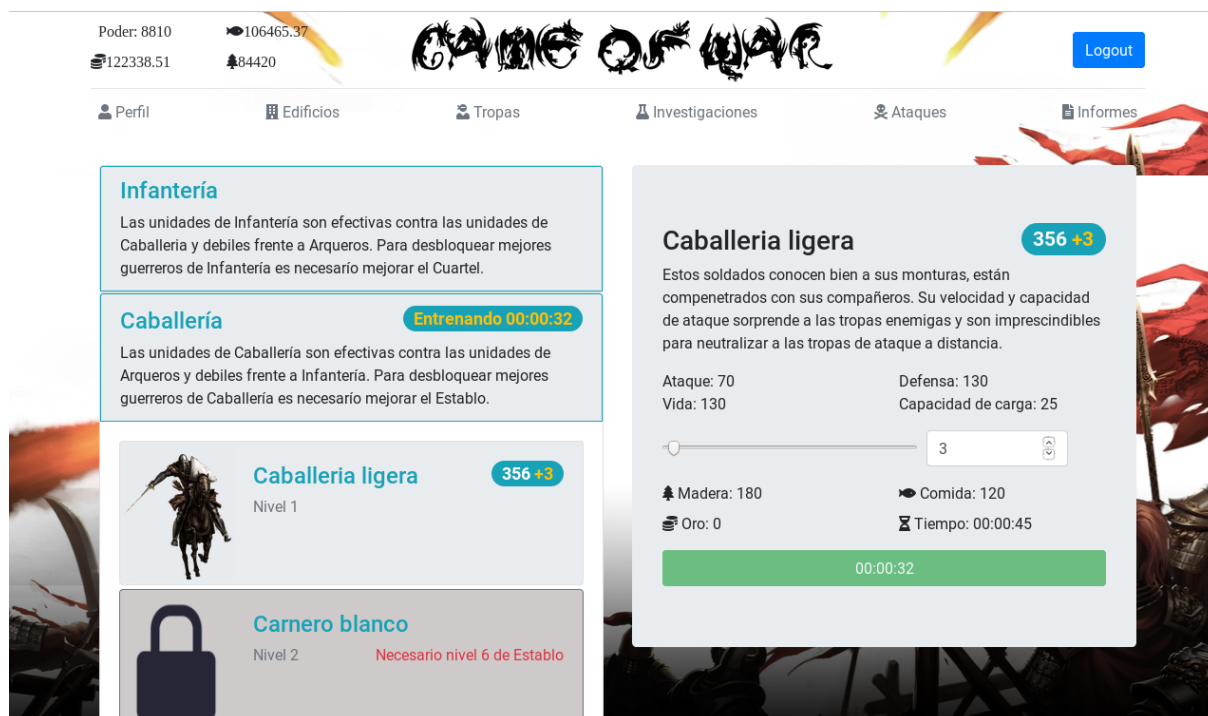


Figura 4.14: Página de tropas

Página de Investigaciones

La tercera opción del menú, Investigaciones, redirige a la ruta `/research`. Funciona de forma muy parecida a los edificios. En este caso es el componente `research-list` quien controla el lado izquierdo de la página, y su hijo, el componente `research`, quien controla el derecho.

En el lado izquierdo aparece una lista con todas las Investigaciones disponibles. Las investigaciones son mejoras, por ejemplo, un incremento en la velocidad con la que se crean las tropas. Al pinchar en una investigación de la lista nos muestra su detalle: para que sirve, que características mejora, el coste en recursos y tiempo. Las investigaciones también pueden estar bloqueadas. Dependiendo del nivel del edificio Academia, habrá unas u otras investigaciones

disponibles para desarrollar. Solo es posible poner una investigación en marcha a la vez y aparece en la parte superior de la lista con una cuenta atrás para su finalización.

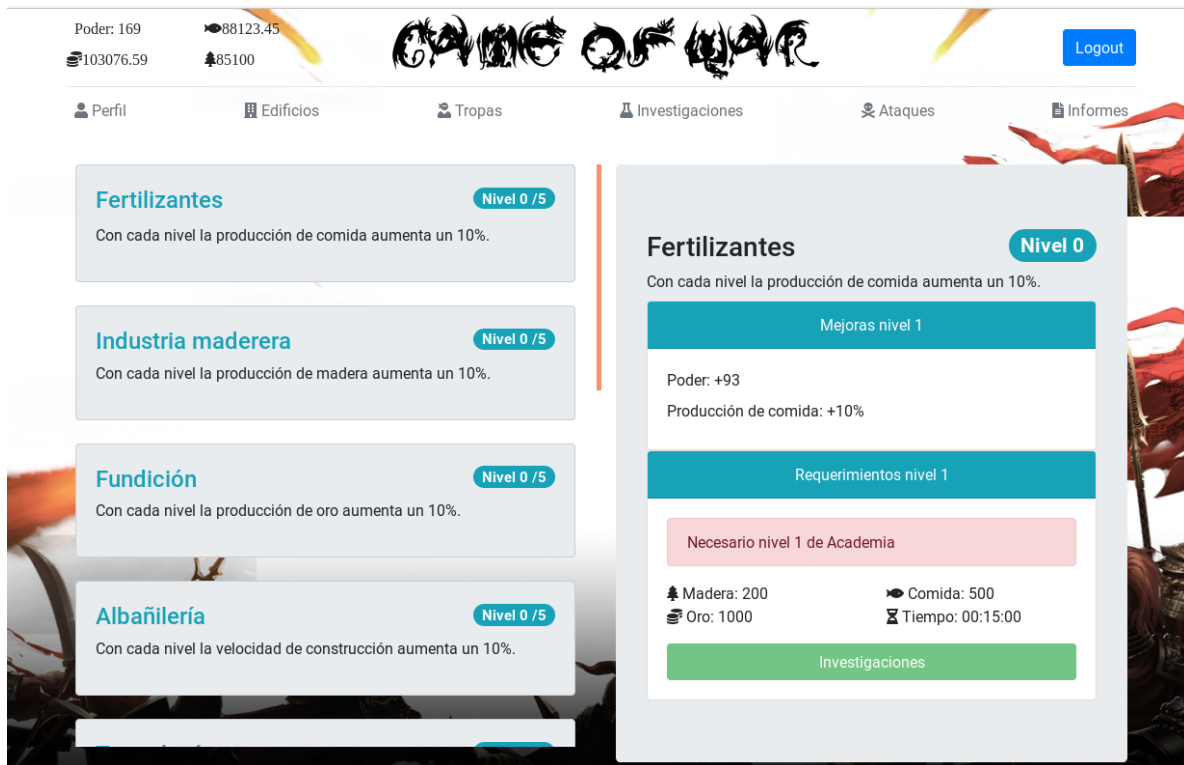


Figura 4.15: Página de investigaciones

Página de ataques

La quinta opción del menú, Ataques, dirige a la ruta /war. Desde aquí podemos atacar a otros jugadores.

En el lado izquierdo aparece una lista con 30 jugadores, rellena con jugadores que te hayan atacado en algún momento y el resto jugadores generados de manera aleatoria.

Para atacar, simplemente hay que seleccionar a que jugador de la lista queremos atacar y las tropas con las que queremos hacerlo.

Al finalizar la batalla se muestra un informe mediante una modal de Bootstrap con el resultado del combate. En ella aparecen dos tablas, una para cada participante, con las tropas iniciales, las tropas muertas, los heridos durante la batalla y los supervivientes. Y en el caso de que el atacante salga victorioso del combate se muestra una tercera tabla más con los recursos robados al defensor, en el caso de que el defensor tuviera recursos y el atacante capacidad de

carga para llevárselos. Esto se detallara más adelante, junto con el sistema de batalla utilizado.



Figura 4.16: Página de ataques

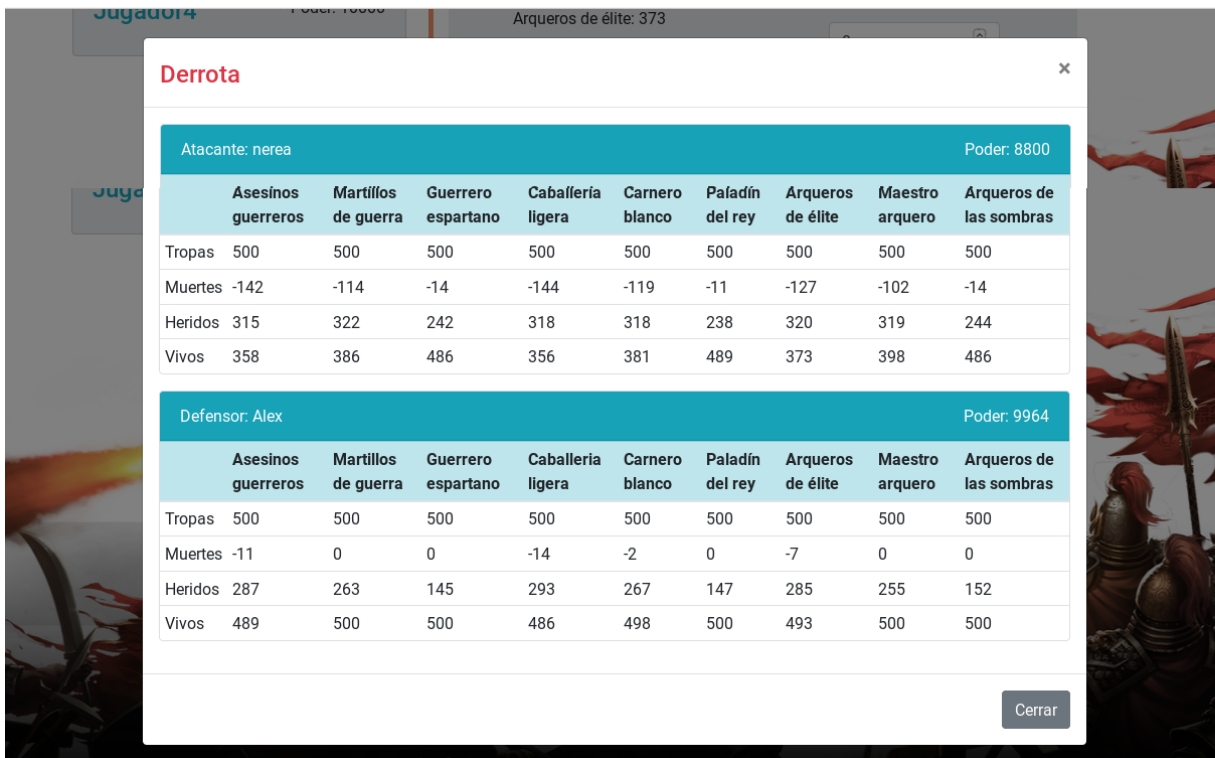


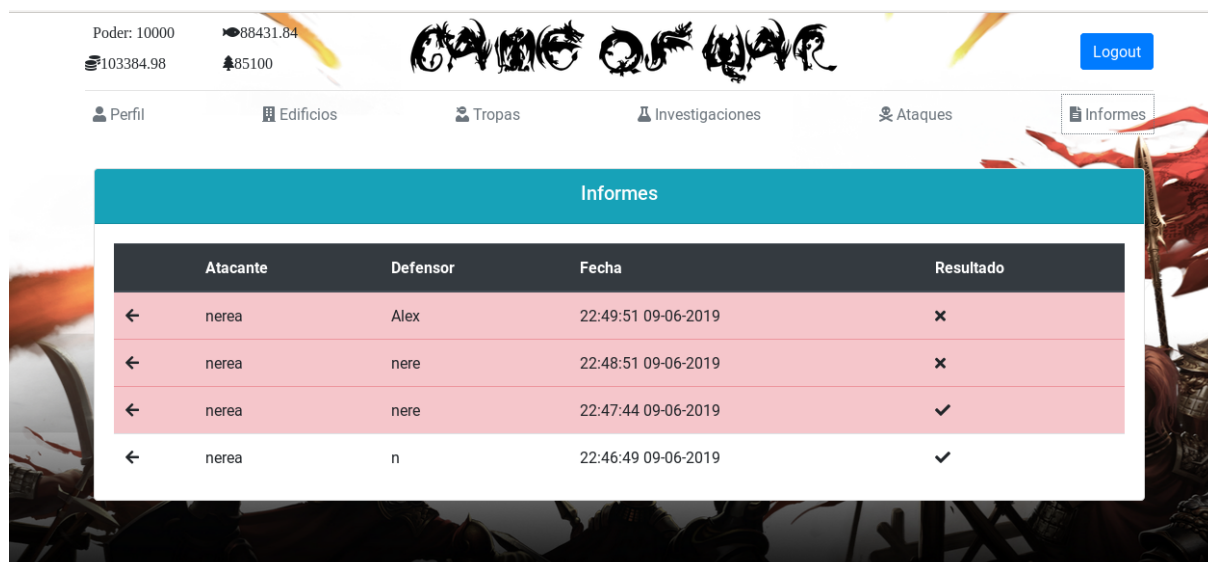
Figura 4.17: Informe

Página de informes

La sexta y última opción del menú, Informes, redirige a la ruta /reports, controlada por el componente reports. Aquí se muestra una tabla con los informes de las batallas realizadas, tanto si hemos sido atacantes como defensores. La lista muestra a ambos jugadores y el rol de cada uno, el resultado de la batalla y la hora y día a la que fue realizado el ataque.

Al pinchar sobre una de las filas de la tabla se nos abre un informe igual que el mostrado tras realizar un ataque: con tablas de tropas una para cada jugador, y los recursos si el atacante sale victorioso. Es decir, en este caso, como no somos siempre el atacante, puede salir una tabla con los recursos perdidos, no solo ganados.

Cuando un informe aún no ha sido “leído” aparece en color rojo para diferenciarlo del resto. Una vez que se ha pinchado en él y aparece el informe cambia de color y se pone blanco.



	Atacante	Defensor	Fecha	Resultado
←	nerea	Alex	22:49:51 09-06-2019	×
←	nerea	nere	22:48:51 09-06-2019	×
←	nerea	nere	22:47:44 09-06-2019	✓
←	nerea	n	22:46:49 09-06-2019	✓

Figura 4.18: Página de informes

4.3.3. Sistema de ataques

El sistema de batalla funciona por rondas. En cada ronda de la batalla hay un ataque y un contraataque, es decir, primero el jugador 1 ataca al jugador 2 y después el jugador 2 pasa a ser el atacante y el jugador 1 el defensor. En las dos partes de la ronda ambos jugadores cuentan con todas las tropas con las que la iniciaron. Los soldados muertos o heridos no se restan hasta terminada la ronda.

El ataque y el contraataque se realiza de la misma manera. Cada tropa del jugador atacante ataca a cada una del defensor, en proporción a las tropas de este, y así son dañadas equitativamente, sin importar el nivel ni el tipo. Esto puede verse en la figura 4.19.

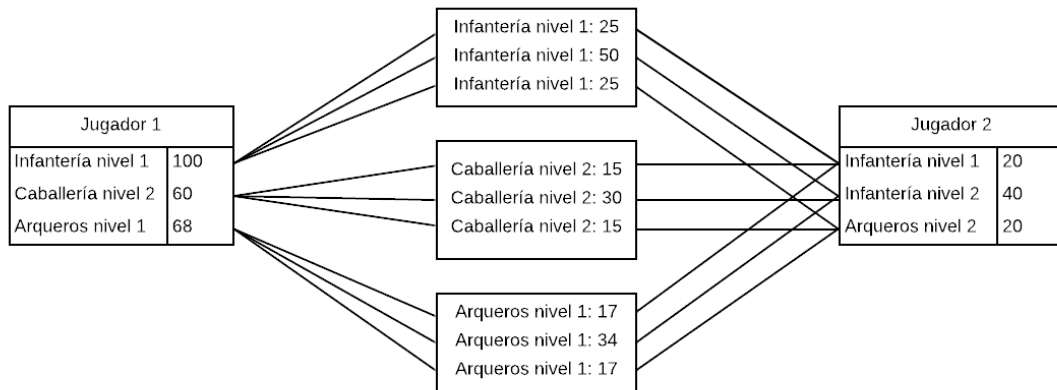


Figura 4.19: Reparto de soldados en un ataque

Este reparto de tropas puede realizarse ya que la fórmula de cálculo del daño no se basa en el número de defensores si no en el número de atacantes. Dividimos por dos razones, primero porque cada tipo de tropa tendrá diferente defensa y, segundo, porque cada tropa tiene un tipo a la que es más vulnerable.

Cálculo del daño

Daño a la vida = $10 * e^{(0,03 * \text{Diferencia de fuerza})}$
 $* (\text{Precisión de la tropa}) * (\text{aleatorio entre 80\% y 120\%})$
 $* (\text{Modificador tipo tropa})$

Diferencia de fuerza = ataque total tropa atacante -
 defensa total tropa defensora

Ataque tropa atacante = ataque tropa * bonus investigaciones de ataque

Ataque tropa atacante = defensa tropa * bonus investigaciones de defensa

Modificador tipo tropa = $1.25 + e^{(\text{nivel tropa atacante} * 0.42)} * 0.1;$

La fórmula se basa en un factor de daño base, es decir, cuando golpeamos, el golpe siempre lleva un mínimo de daño, que se verá incrementado en función del ataque de nuestras tropas o

se verá reducido debido a la defensa del defensor. El factor mínimo escogido es 10, modificarlo haría que variara el número de rondas en la batalla.

La exponencial se utiliza para calcular el aumento del daño en función de la diferencia de fuerzas. La diferencia de fuerza es el ataque total de la tropa atacante menos la defensa total de la tropa defensora.

El ataque total está formado por el ataque de la tropa en cuestión por el porcentaje de mejora de ataque conseguido a través de las investigaciones. La defensa total se calcula de la misma manera.

El modificador 0.03 se ha elegido para ajustar el crecimiento del daño según la diferencia de fuerza. Si este valor fuera más grande, con poca diferencia de fuerzas el daño podría ser exagerado. Cuanto más pequeño sea este valor mayor margen tendremos en diferencia de fuerzas antes de que el valor se dispare de forma aplastante (Es lógico que una tropa de 250 de ataque aplaste sin problemas a otra de 80 defensa, incluso de forma exagerada... pero si no, se puede controlar cambiando este parámetro).

La precisión de la tropa es un número entre 0 y 1, que indica qué puntería tiene la tropa. A mayor nivel mejor puntería.

También incluimos un factor suerte, que puede hacer bajar el rendimiento de la tropa o aumentarlo para causar más daño, en función de un número aleatorio, entre 80 % y 120 %.

Cada tipo de tropa es mejor o peor frente a otros tipos de tropas y esto mejora u empeora el ataque según el cálculo puesto, que va acorde con el crecimiento. La relación sería:

- La infantería es fuerte contra la caballería y débil contra los arqueros.
- La caballería es fuerte contra los arqueros y débil contra la infantería.
- La arqueros son fuerte contra la infantería y débil contra la caballería.

El jugador defensor de la batalla (no el de la ronda) cuenta además con un suplemento de defensa.

Cálculo de Muertes

El cálculo de las muertes se realiza de la siguiente manera:

Muertes = N° unidades tropa atacante * Daño a la vida / vida tropa defensora

Pero en un ataque puede haber muertos y/o heridos.

Los heridos son un porcentaje de los muertos, calculados en función del poder de la defensa y la vida de las tropas. Cuanto mayor sea la defensa más proporción de heridos se conseguirá. Si la defensa supera al ataque no habrá muertos, mientras que un ataque demasiado poderoso podría provocar cero heridos, todo muertos.

Como se ha dicho, al finalizar la ronda se restan los muertos a las tropas de cada jugador. Pero también se restan temporalmente los heridos, no podrán participar en las rondas restantes. Al finalizar el combate cada jugador recuperará las tropas heridas.

El número máximo de rondas viene dado, bien por un valor fijo de 250 rondas, bien por un límite de tropas perdidas del 85 % por parte de alguno de los jugadores.

Al finalizar el combate se restan las tropas totales muertas de cada jugador y el poder perdido con esas tropas muertas. En el caso de que el jugador atacante salga victorioso se realiza el cálculo de los posibles recursos robados al enemigo.

Cálculo de recursos ganados

Para realizar el cálculo se tiene en cuenta dos cosas, la cantidad de recursos que puede perder el jugador defensor y la cantidad de recursos que puede transportar el jugador atacante:

Los recursos que puede perder el defensor = La cantidad de recursos que tiene el defensor - la cantidad de recursos protegidos por el edificio Almacén

Esto se realiza para cada tipo de recurso. A mayor nivel del edificio en cuestión mayor número de recursos protegidos. Por otro lado, se calcula:

El número de recursos que puede transportar el atacante = número de tropas sanas (sin heridos) * la capacidad de carga de los diferentes tipos de tropas.

Si la capacidad de carga del atacante es mayor que los recursos que puede perder el defensor se llevará todos ellos. En caso contrario, se llevará de cada tipo de recurso una cantidad realizada de manera proporcional a la cantidad de ese recurso que hay disponible y de manera que la suma de los tres tipos iguale el máximo de su capacidad de carga. Es decir, por ejemplo:

Si el defensor dispone de:

200000 de comida , 100000 de madera y 50000 de oro

Y la capacidad máxima de carga es 100000 la cantidad robada será:

57142.86 de comida, 28571.43 de madera y 14285.71 de oro

Capítulo 5

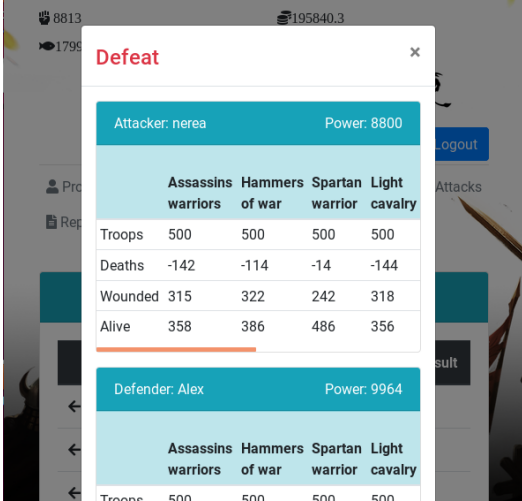
Resultados

El resultado final del proyecto es satisfactorio. El presente proyecto pretendía crear un juego de estrategia militar, cosa que se ha logrado.

El juego tiene una interfaz sencilla y se adapta a los diferentes tamaños de pantallas, como se había dicho. Esto puede verse en las figuras 5.1 y 5.2.



Figura 5.1: Diseño responsive 1



Attacker: nerea		Power: 8800			
	Assassins warriors	Hammers of war	Spartan warrior	Light cavalry	
Troops	500	500	500	500	
Deaths	-142	-114	-14	-144	
Wounded	315	322	242	318	
Alive	358	386	486	356	
Defender: Alex		Power: 9964			
	Assassins warriors	Hammers of war	Spartan warrior	Light cavalry	
Troops	500	500	500	500	

Figura 5.2: Diseño responsive 2

Se da la posibilidad de visualizar la aplicación tanto en inglés como en español. Además es posible añadir nuevos lenguajes simplemente añadiendo un fichero JSON, como el explicado en el capítulo 4, e iguales a los que ya hay, con los datos de las plantillas y añadiendo a la base de datos las traducciones de los edificios, investigaciones y demás características del juego.

El juego funciona en tiempo real, como se pretendía. Las investigaciones, construcciones y creaciones de tropas siguen avanzando aún cuando el jugador no está activo, al igual que sus recursos aumentan y es posible que reciba ataques durante ese tiempo.

El sistema de batallas parte de la relación entre el daño y la diferencia de fuerza de los jugadores. A ello se le han añadido los modificadores explicados en el capítulo 4: un porcentaje de suerte, puntería de las tropas y ventaja entre tipos de tropas. No se ha querido dejar que los jugadores pierdan todas sus tropas en un solo ataque, de modo que se han incluido los heridos, y se finalizan los combates antes de llegar a un mínimo de tropas. Parece que las batallas realizadas dan valores equilibrados y no muy excesivos.

Capítulo 6

Conclusiones

6.1. Aplicación de lo aprendido

En la realización de este proyecto he podido poner en práctica mis conocimientos adquiridos durante el grado sobre todo en cuestiones de bases de datos relacionales, HTML, CSS3 y JavaScript. Las demás asignaturas relacionadas con la programación me han sido útiles de igual modo, ya que aún tratándose de lenguajes o tecnologías diferentes, aportan conocimientos sobre el funcionamiento y la forma de desarrollar una aplicación.

La asignatura más relacionada en cuanto a tecnologías es Desarrollo de aplicaciones telemáticas. Es en esta asignatura donde adquirí mis principales conocimientos sobre JavaScript, HTML5, CSS3 y distintas APIs de Google.

En Ingeniería de sistemas de información aprendí a construir una aplicación completa usando herramientas para el desarrollo en equipo. Y adquirí mis conocimientos sobre bases de datos.

Otras asignaturas relacionadas son también Arquitectura de redes de ordenadores, Sistemas telemáticos y Servicios y aplicaciones telemáticas. En ellas aprendí lo que era una aplicación REST, el funcionamiento de los protocolos y peticiones HTTP, entre otras cosas.

6.2. Lecciones aprendidas

Ha supuesto un reto enfrentarse a un proyecto de esta envergadura y más aún tratándose, en su mayoría, de tecnologías que desconocía y que nunca había utilizado, como son Angular, junto con el lenguaje TypeScript, Node.js y Bootstrap.

He podido aprender mucho sobre el desarrollo de aplicaciones escalables y de programación asíncrona, sin bloqueos, que me ha proporcionado también muchos quebraderos de cabeza.

A pesar de estar basado en JavaScript y ser similar, TypeScript junto con Angular suponen una manera totalmente diferente de programar de lo que conocía hasta ahora con JavaScript. También puedo decir que Angular facilita el mantenimiento y desarrollo de una aplicación de gran envergadura.

Por otro lado, con Bootstrap he aprendido nuevas formas de mejorar el aspecto visual de las aplicaciones web y de darles un diseño responsive de forma más sencilla.

También he afianzado mis conocimientos sobre bases de datos, cosa que he utilizado mucho para la realización del proyecto y que hasta ahora apenas las había empezado a usar.

Para la realización de esta memoria he utilizado otra herramienta que desconocía, Latex, que ha resultado muy útil para conseguir una buena presentación de la memoria.

Además de todo esto, durante el desarrollo utilicé otras herramientas que tampoco conocía: Postman, que me ha ayudado a comprobar el funcionamiento de mi servidor mediante la generación de peticiones HTTP, y Morgan, un middleware para Node.js que sirve para poder ver en la consola del servidor las peticiones HTTP que este va recibiendo.

6.3. Trabajos futuros

Esta aplicación es solo el comienzo de lo que podría llegar a ser este juego. A un software siempre se le pueden realizar mejoras o añadir nuevas funcionalidades, y como tal, este juego no se queda atrás.

Para empezar, la parte más importante que se podría añadir a este juego sería una interfaz visual atractiva, que incluyera mapas e imágenes completas de ciudades, batallas que pudieran visualizarse, etc. Y en estos mapas no tendrían por qué contener solo las ciudades de los jugadores, o limitarnos a movernos entre ellas, podría haber otros elementos como por ejemplo mares y barcos para poder cruzar, etc.

También se podrían añadir opciones para permitir la iteración y colaboración con otros jugadores, como poder comunicarte con ellos, intercambios de materiales e incluso tropas, la realización de torneos o eventos para que los jugadores compitieran.

Las batallas se podrían modificar de forma que el jugador tomara más partido en ellas, no

solo mediante la elección de las tropas adecuadas, si no por ejemplo eligiendo el orden exacto en el que quiere que ataquen sus tropas, o el orden de los objetivos, etc, lo cual modificaría por completo el resultado de una batalla aún con las mismas tropas. También se puede añadir la realización de espionajes, de ataques conjuntos, etc.

Habría multitud de nuevas posibilidades.

Como ya he dicho también, el acceso de los jugadores no está limitado, cada jugador puede tener el número de cuentas que desee, esto podría modificarse y establecer un límite de cuenta por persona.

Apéndice A

Manual de instalación

Se puede descargar o clonar el proyecto desde su página de Github:

```
http://github.com/nereac/projectGameOfWar
```

Para utilizar el juego hay que seguir el siguiente proceso:

Instalación de Node.js y NPM

Para instalar la última versión de Node.js hay que ejecutar, en una terminal, los siguientes comandos:

```
sudo apt install curl
curl -sL https://deb.nodesource.com/setup_11.x -o nodesource_setup.sh
sudo bash nodesource_setup.sh
sudo apt-get install Node.js
```

Instalación de Angular CLI

```
sudo npm install -g @angular/cli
```

Instalación de MySQL

```
sudo apt install mysql-server
sudo mysql_secure_installation
```

Con este último comando se entra en una pantalla de configuraciones. Seleccione la opción 0 e introduzca la contraseña 'dbpro2019'. A continuación pulse Y y luego Enter para el resto de configuraciones.

Después es necesario ejecutar los siguientes comandos:

```
sudo mysql
ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password
  BY 'dbpro2019';
FLUSH PRIVILEGES;
exit
```

Se puede elegir una contraseña diferente pero habría que modificar el fichero `server/src/keys.ts`, donde está incluida la contraseña.

Creación de la base de datos

Es necesario crear y rellenar la base de datos con los contenidos del juego. Partiendo del directorio del proyecto:

```
cd database
mysql -u root -p // introducimos contraseña, dbpro2019.
source database.sql
```

Servidores

Lanzar el servidor de Express ejecutando el siguiente comando en un terminal:

```
npm run dev
```

Lanzar el servidor de Angular ejecutando el siguiente comando en un terminal diferente:

```
ng serve
```

Acceso

La url para acceder al juego es la siguiente:

```
http://localhost:4200
```


Bibliografía

- [1] Información sobre la librería de Angular ngx-translate.
<https://github.com/ngx-translate/core>.
- [2] Página de Expressjs.
<https://expressjs.com/es/>.
- [3] Página oficial de Angular.
<https://angular.io/>.
- [4] Página oficial de Angular Cli.
<https://cli.angular.io/>.
- [5] Página oficial de Bootstrap.
<http://getbootstrap.com/>.
- [6] Página oficial de Momentjs.
<https://momentjs.com/>.
- [7] Página oficial de Mysql.
<https://www.mysql.com/>.
- [8] Página oficial de Node.js.
<https://nodejs.org/es/>.
- [9] Página oficial de NPM.
<https://www.npmjs.com/>.
- [10] Página sobre CSS3.
http://www.w3schools.com/css/css3_intro.asp/.

[11] Página sobre HTML.

https://www.w3schools.com/html/html5_intro.asp.

[12] Página sobre HTML.

<https://html.spec.whatwg.org/multipage/>.

[13] Página sobre Json Web Token.

<https://jwt.io/>.

[14] Página oficial de Typescript.

<https://www.typescriptlang.org/>.